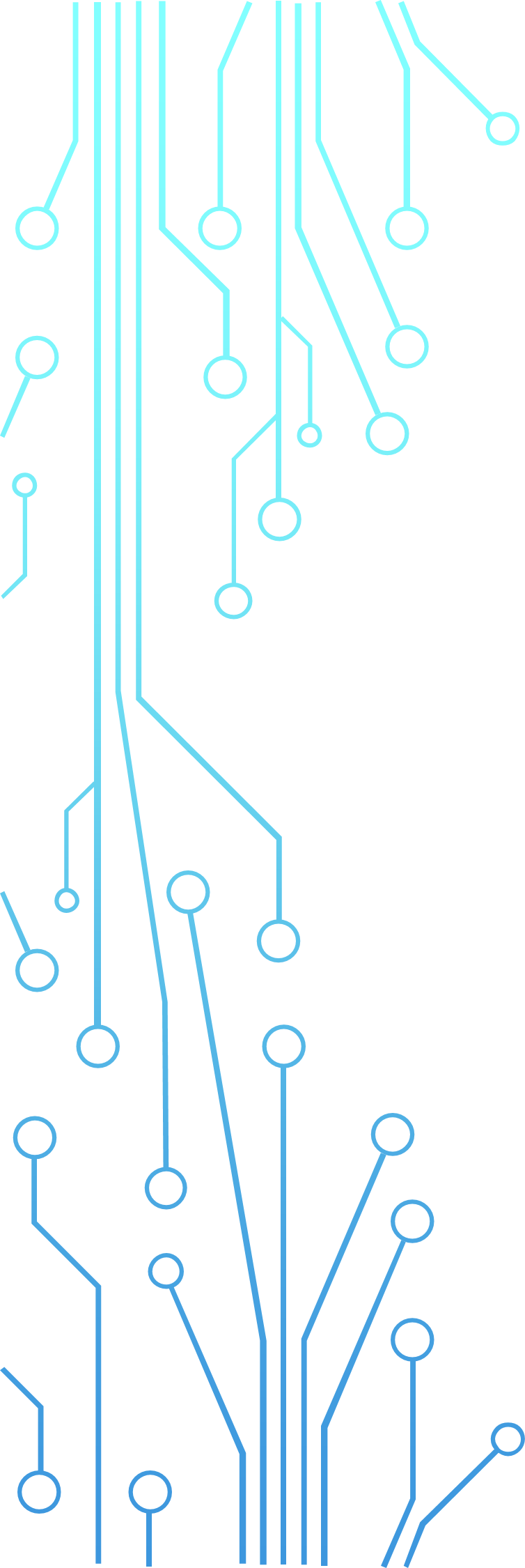
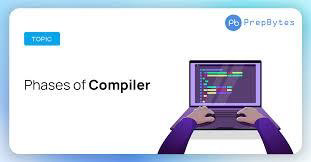
**CSA1452 –COMPILER DESIGN**

**FOR DYNAMIC LINKING-slot A**



**Guided By, Project By,**

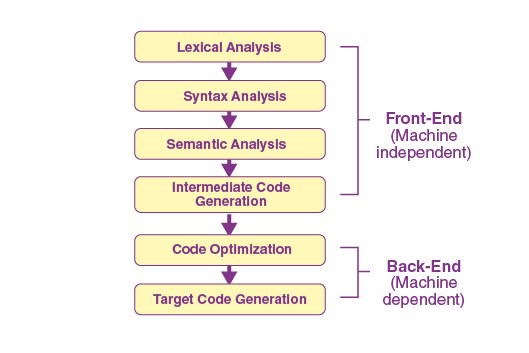
# R. S A IR A M. A M. SIVASATHWIK

*(Course Faculty) (192211146)*

COMPILER DESIGN Computer Science & Engineering

SSE, SIMATS SSE, SIMATS

# AIM

* The different phases of compiler design include lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization, and code generation.
* Each phase plays a crucial role in the compilation process.
* The goal is to produce an executable program that can run on the target hardware. 

# ABSTRACT

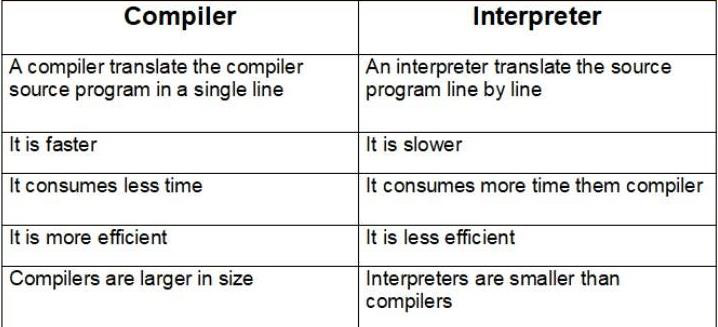
* The phases of compiler design consist of several steps that transform source code into executable machine code.
* These phases include lexical analysis, which breaks the code into tokens.
* syntax analysis, which checks the code's grammar and structure.
* semantic analysis, which ensures the code's meaning is correct.
* code generation, which produces the target machine code.
* optimization, which improves the efficiency of the generated code.
* Each phase plays a crucial role in the compilation process, ultimately enabling programs to be executed on a computer.

# COMPILER

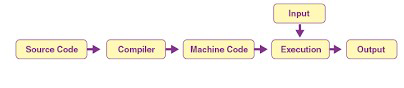
* compiler is a sophisticated tool that translates high-level programming language code into machine code or another form of code that can be executed by a computer.
* It plays a crucial role in software development by enabling programmers to write code in a human-readable format while producing efficient and reliable executable programs.



## COMPILER VS INTERPRETER



# INTRODUCTION



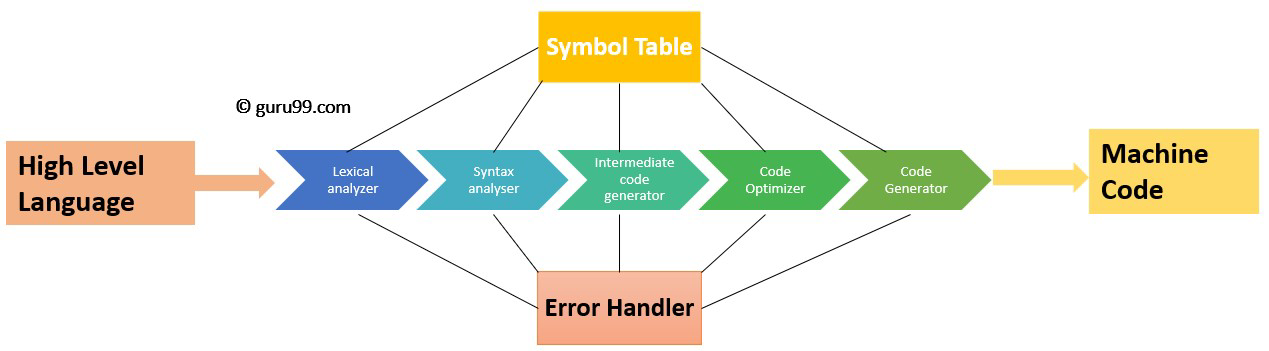
* Analysis of compiler design typically covers the foundational concepts and techniques involved in building compilers.
* Compiler design is the process of creating a software system that translates source code written in a high-level programming language into equivalent machine code .
* It involves a series of stages aimed at analyzing, transforming, and generating code efficiently.

# LITERATURE SURVEY

**AUTHOR-1:-**

* **TITLE:-**several types of automated tools used in various **phases** of a **compiler design.**
* **YEAR:-2022**
* **AUTH0R 2:-**
* **TITLE:-**transformation **phase** is presented in much detail in the volume **Analysis**
* **YEAR:-2013**
* **AUTHOR 3:-**
* **TITLE:-**Advanced **compiler design** implementation.
* **YEAR:-1997**

# ARCHITECTURE



**PHASE 1:-LEXICAL ANALYSIS (SCANNING)**:

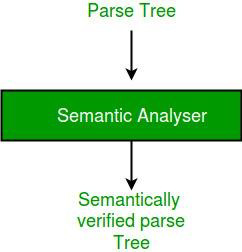
* The input to this phase is the source code.
* It breaks the source code into a sequence of lexemes or tokens.
* Commonly performed by a lexer or scanner.
* Discards white spaces and comments.

**PHASE 2:-SYNTAX ANALYSIS (PARSING)**:

* This phase verifies that the tokens produced by the lexical analysis are arranged in a grammatically correct structure according to the language's syntax rules.
* Generates a parse tree or abstract syntax tree (AST).
* Commonly performed by a parser.

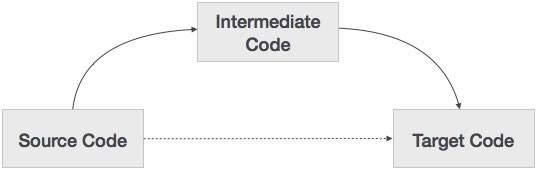
**PHASE 3:-SEMANTIC ANALYSIS**:

* Checks the meaning of the statements in the source code beyond the grammar.
* Ensures that the code adheres to the semantic rules of the programming language.
* Performs type checking, checks for variable usage errors, etc.
* May involve symbol table management.



**PHASE 4:-INTERMEDIATE CODE GENERATION**:

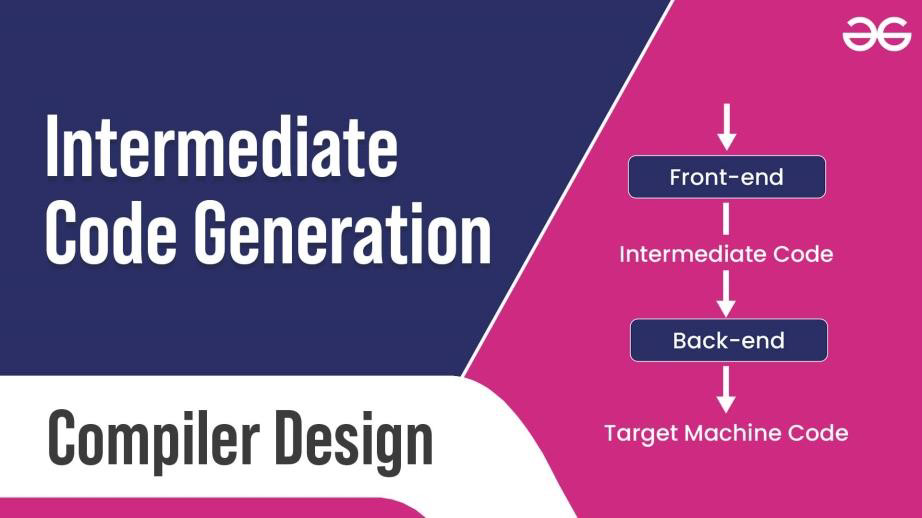
* Converts the parse tree or AST into an intermediate representation (IR) or intermediate code.
* This representation is independent of the source language but easier to translate into the target machine code.
* May involve optimizations at this stage (e.g., constant folding, common subexpression elimination).



**PHASE 5:-OPTIMIZATION**:

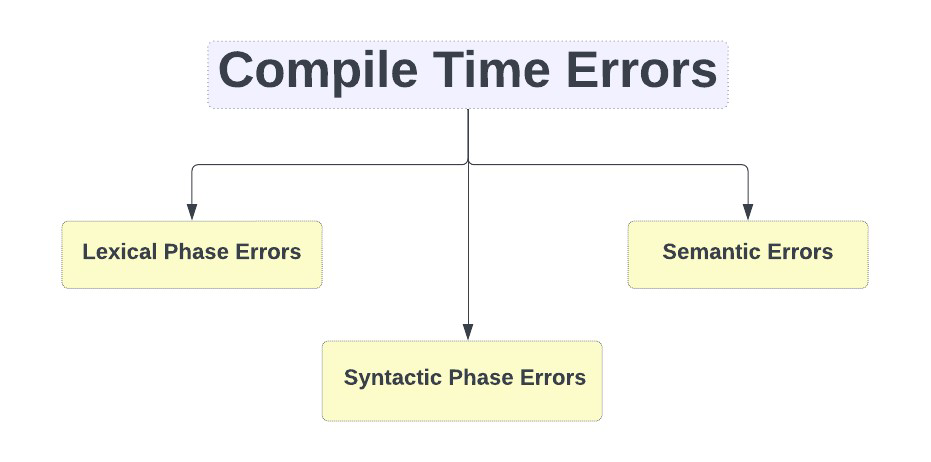
* Performs various transformations on the intermediate code to improve the efficiency of the generated code.
* Includes optimizations like loop optimization, dead code elimination, register allocation, etc.
* Optimization techniques vary widely and can be numerous.

**PHASE 6:-CODE GENERATION**:

* Translates the optimized intermediate code into the target machine code or another language.
* May involve instruction selection, scheduling, and other low-level optimizations.
* Output can be machine code, assembly code, or another high-level language. 

**PHASE 7:-ERROR HANDLING**:

* Detects and reports errors found during the compilation process.
* Errors can be lexical, syntactical, semantic, or related to code generation.
* Provides informative error messages to aid programmers in debugging.



**FRONT END AND BACK END**

* The phases of a compiler are collected into front end and back end.

# •FRONT END:-

* Front-end compilers, also known as front-end tools**.**
* responsible for the initial stages of the compilation process, handling tasks such as lexical analysis, syntax analysis, and semantic analysis.
* These components play a crucial role in translating high-level source code into an intermediate representation (IR)

## •BACK END:-

* In the context of compilers, the "back end" refers to the latter stages of the compilation process, specifically focusing on the generation of target code.
* the back end of a compiler is responsible for translating the optimized intermediate representation.